

# Package ‘gDRutils’

May 15, 2024

**Type** Package

**Title** A package with helper functions for processing drug response data

**Version** 1.2.0

**Date** 2024-04-15

**Description** This package contains utility functions used throughout the gDR platform to fit data, manipulate data, and convert and validate data structures. This package also has the necessary default constants for gDR platform. Many of the functions are utilized by the gDRcore package.

**License** Artistic-2.0

**LazyLoad** yes

**Depends** R (>= 4.2)

**Imports** BiocParallel, BumpyMatrix, checkmate, data.table, drc, jsonlite, jsonvalidate, methods, MultiAssayExperiment, S4Vectors, stats, stringr, SummarizedExperiment

**Suggests** BiocManager, BiocStyle, futile.logger, gDRstyle (>= 1.1.5), gDRtestData (>= 1.1.10), IRanges, knitr, lintr, purrr, qs, rcmdcheck, rmarkdown, testthat, tools, yaml

**URL** <https://github.com/gdrplatform/gDRutils>,  
<https://gdrplatform.github.io/gDRutils/>

**BugReports** <https://github.com/gdrplatform/gDRutils/issues>

**biocViews** Software, Infrastructure

**VignetteBuilder** knitr

**ByteCompile** TRUE

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.1

**SwitchrLibrary** gDRutils

**DeploySubPath** gDRutils

**Encoding** UTF-8

**git\_url** <https://git.bioconductor.org/packages/gDRutils>

**git\_branch** RELEASE\_3\_19

**git\_last\_commit** d1f7215

**git\_last\_commit\_date** 2024-04-30

**Repository** Bioconductor 3.19

**Date/Publication** 2024-05-15

**Author** Bartosz Czech [aut] (<<https://orcid.org/0000-0002-9908-3007>>),  
 Arkadiusz Gladki [cre, aut] (<<https://orcid.org/0000-0002-7059-6378>>),  
 Aleksander Chlebowski [aut],  
 Marc Hafner [aut] (<<https://orcid.org/0000-0003-1337-7598>>),  
 Pawel Piatkowski [aut],  
 Dariusz Scigocki [aut],  
 Janina Smola [aut],  
 Sergiu Mocanu [aut],  
 Allison Vuong [aut]

**Maintainer** Arkadiusz Gladki <[gladki.arkadiusz@gmail.com](mailto:gladki.arkadiusz@gmail.com)>

## Contents

gDRutils-package . . . . .	4
.convert_mae_summary_to_json . . . . .	5
.convert_norm_specific_metrics . . . . .	5
.set_invalid_fit_params . . . . .	6
addClass . . . . .	6
aggregate_assay . . . . .	7
apply_bumpy_function . . . . .	8
assert_choices . . . . .	9
average_biological_replicates_dt . . . . .	9
cap_xc50 . . . . .	10
convert_colData_to_json . . . . .	11
convert_combo_data_to_dt . . . . .	12
convert_combo_field_to_assay . . . . .	13
convert_mae_assay_to_dt . . . . .	13
convert_mae_to_json . . . . .	15
convert_metadata_to_json . . . . .	15
convert_rowData_to_json . . . . .	16
convert_se_assay_to_dt . . . . .	17
convert_se_to_json . . . . .	18
demote_fields . . . . .	19
df_to_bm_assay . . . . .	20
extend_normalization_type_name . . . . .	20
fit_curves . . . . .	21
flatten . . . . .	22
gen_synthetic_data . . . . .	23
geometric_mean . . . . .	24
get_assay_names . . . . .	24

get_combo_assay_names . . . . .	25
get_combo_base_assay_names . . . . .	26
get_combo_col_settings . . . . .	26
get_combo_excess_field_names . . . . .	27
get_combo_score_assay_names . . . . .	27
get_combo_score_field_names . . . . .	28
get_default_identifiers . . . . .	28
get_duplicated_rows . . . . .	29
get_env_assay_names . . . . .	29
get_expect_one_identifiers . . . . .	30
get_experiment_groups . . . . .	31
get_identifiers_dt . . . . .	31
get_ids_synonyms . . . . .	32
get_iso_colors . . . . .	33
get_MAE_identifiers . . . . .	33
get_non_empty_assays . . . . .	34
get_optional_coldata_fields . . . . .	34
get_optional_rowdata_fields . . . . .	35
get_required_identifiers . . . . .	35
get_synthetic_data . . . . .	36
get_testdata . . . . .	36
get_testdata_codilution . . . . .	37
get_testdata_combo . . . . .	37
headers . . . . .	38
identifiers . . . . .	38
identify_unique_se_metadata_fields . . . . .	40
is_any_exp_empty . . . . .	40
is_exp_empty . . . . .	41
is_mae_empty . . . . .	42
logisticFit . . . . .	42
loop . . . . .	44
MAEapply . . . . .	45
mcolData . . . . .	46
merge_assay . . . . .	46
merge_metadata . . . . .	47
merge_SE . . . . .	48
modifyData . . . . .	49
mrowData . . . . .	50
predict_conc_from_efficacy . . . . .	51
predict_efficacy_from_conc . . . . .	52
prettify_flat_metrics . . . . .	53
promote_fields . . . . .	54
refine_coldata . . . . .	55
refine_rowdata . . . . .	55
rename_bumpy . . . . .	56
rename_DFrame . . . . .	57
set_constant_fit_params . . . . .	57
SE_metadata . . . . .	58

shorten_normalization_type_name . . . . .	59
split_SE_components . . . . .	60
standardize_mae . . . . .	61
standardize_se . . . . .	62
strip_first_and_last_char . . . . .	62
update_env_idfs_from_mae . . . . .	63
update_idfs_synonyms . . . . .	63
validate_dimnames . . . . .	64
validate_identifiers . . . . .	64
validate_json . . . . .	65
validate_MAE . . . . .	66
validate_mae_with_schema . . . . .	67
validate_SE . . . . .	67
validate_se_assay_name . . . . .	68

## Index 70

---

gDRutils-package	<i>gDRutils: A package with helper functions for processing drug response data</i>
------------------	--

---

## Description

This package contains utility functions used throughout the gDR platform to fit data, manipulate data, and convert and validate data structures. This package also has the necessary default constants for gDR platform. Many of the functions are utilized by the gDRcore package.

## Value

package help page

## Note

To learn more about functions start with `help(package = "gDRutils")`

## Author(s)

**Maintainer:** Arkadiusz Gladki <gladki.arkadiusz@gmail.com> ([ORCID](#))

Authors:

- Bartosz Czech ([ORCID](#))
- Aleksander Chlebowski
- Marc Hafner ([ORCID](#))
- Pawel Piatkowski
- Dariusz Scigocki
- Janina Smola
- Sergiu Mocanu
- Allison Vuong

### See Also

Useful links:

- <https://github.com/gdrplatform/gDRutils>
- <https://gdrplatform.github.io/gDRutils/>
- Report bugs at <https://github.com/gdrplatform/gDRutils/issues>

---

`.convert_mae_summary_to_json`

*Create JSON document with MAE summary*

---

### Description

Create JSON document with MAE summary, currently only experiment names

### Usage

```
.convert_mae_summary_to_json(mae)
```

### Arguments

`mae` MultiAssayExperiment object.

### Value

String representation of a JSON document.

---

`.convert_norm_specific_metrics`

*This function change raw names of metric from long format table into more descriptive names in the wide format table. It works for metrics: `colnames(get_header("metrics_names"))`*

---

### Description

This function change raw names of metric from long format table into more descriptive names in the wide format table. It works for metrics: `colnames(get_header("metrics_names"))`

### Usage

```
.convert_norm_specific_metrics(x, normalization_type)
```

```
.set_invalid_fit_params
```

*Set fit parameters for an invalid fit.*

---

**Description**

Set fit parameters for an invalid fit.

**Usage**

```
.set_invalid_fit_params(out, norm_values)
```

**Arguments**

out	Named list of fit parameters.
norm_values	Numeric vector used to estimate an $\chi^2$ value.

**Value**

Modified named list of fit parameters.

**Examples**

```
.set_invalid_fit_params(list(), norm_values = rep(0.3, 6))
```

---

```
addClass
```

*add arbitrary S3 class to an object*

---

**Description**

Modify an object's class attribute.

**Usage**

```
addClass(x, newClass)
```

**Arguments**

x	an object
newClass	character string; class to be added

**Details**

This is a simple convenience function that adds an item to the class attribute of an object so that it can be dispatched to a proper S3 method. This is purely for code clarity, so that individual methods do not clutter the definitions of higher order functions.

**Value**

The same object with an added S3 class.

**Examples**

```
addClass(data.table::data.table(), "someClass")
```

---

aggregate_assay	<i>Aggregate a BumpyMatrix assay by a given aggregation function.</i>
-----------------	---

---

**Description**

Aggregation can only be performed on nested variables.

**Usage**

```
aggregate_assay(asy, by, FUN)
```

**Arguments**

asy	A BumpyMatrix object.
by	Character vector of the nested fields to aggregate by.
FUN	A function to use to aggregate the data.

**Value**

A BumpyMatrix object aggregated by FUN.

**Examples**

```
mae <- get_synthetic_data("finalMAE_small")
se <- mae[[1]]
assay <- SummarizedExperiment::assay(se)
aggregate_assay(assay, FUN = mean, by = c("Barcode"))
```

---

apply\_bumpy\_function *Apply a function to every element of a bumpy matrix.*

---

### Description

Apply a user-specified function to every element of a bumpy matrix.

### Usage

```
apply_bumpy_function(
  se,
  FUN,
  req_assay_name,
  out_assay_name,
  parallelize = FALSE,
  ...
)
```

### Arguments

se	A SummarizedExperiment object with bumpy matrices.
FUN	A function that will be applied to each element of the matrix in assay req_assay_name. Output of the function must return a data.table.
req_assay_name	String of the assay name in the se that the FUN will act on.
out_assay_name	String of the assay name that will contain the results of the applied function.
parallelize	Logical indicating whether or not to parallelize the computation.
...	Additional args to be passed to teh FUN.

### Value

The original se object with a new assay, out\_assay\_name.

### Examples

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
FUN <- function(x) {
  data.table::data.table(Concentration = x$Concentration, CorrectedReadout = x$CorrectedReadout)
}
apply_bumpy_function(
  se,
  FUN = FUN,
  req_assay_name = "RawTreated",
  out_assay_name = "CorrectedReadout"
)
```



---

assert_choices	<i>assert choices</i>
----------------	-----------------------

---

**Description**

assert choices

**Usage**

```
assert_choices(x, choices, ...)
```

**Arguments**

x	charvec expected subset
choices	charvec reference set
...	Additional arguments to pass to <code>checkmate::test_choice</code>

**Value**

NULL

**Examples**

```
assert_choices("x", c("x", "y"))
```

---

average_biological_replicates_dt	<i>Average biological replicates.</i>
----------------------------------	---------------------------------------

---

**Description**

Average biological replicates on the data table side.

**Usage**

```
average_biological_replicates_dt(  
  dt,  
  var,  
  pidfs = get_prettified_identifiers(),  
  fixed = TRUE,  
  geometric_average_fields = get_header("metric_average_fields")$geometric_mean  
)
```

**Arguments**

dt	data.table with Metric data
var	String representing additional metadata of replicates
pidfs	list of prettified identifiers
fixed	Flag should be add fix for -Inf in geometric mean.
geometric_average_fields	Character vector of column names in dt to take the geometric average of.

**Value**

data.table without replicates

**Examples**

```
dt <- data.table::data.table(a = c(1:10, 1),
  b = c(rep("drugA", 10), rep("drugB", 1)))
average_biological_replicates_dt(dt, var = "a")
```

---

cap\_xc50

*Cap XC50 value.*

---

**Description**

Set IC50/GR50 value to Inf or -Inf based on upper and lower limits.

**Usage**

```
cap_xc50(xc50, max_conc, min_conc = NA, capping_fold = 5)
```

**Arguments**

xc50	Numeric value of the IC50/GR50 to cap.
max_conc	Numeric value of the highest concentration in a dose series used to calculate the xc50.
min_conc	Numeric value of the lowest concentration in a dose series used to calculate the xc50. If NA (default), using max_conc/1e5 instead.
capping_fold	Integer value of the fold number to use for capping. Defaults to 5.

**Details**

Note: xc50 and max\_conc should share the same units. Ideally, the lower\_cap should be based on the lowest tested concentration. However, since we don't record that, it is set 5 orders of magnitude below the highest dose.

**Value**

Capped IC50/GR50 value.

**Examples**

```
cap_xc50(xc50 = 1, max_conc = 2)
cap_xc50(xc50 = 2, max_conc = 5, min_conc = 1)
cap_xc50(xc50 = 26, max_conc = 5, capping_fold = 5)
```

---

convert\_colData\_to\_json

*Convert colData to JSON*

---

**Description**

Convert colData to JSON format for elasticsearch indexing.

**Usage**

```
convert_colData_to_json(
  cdata,
  identifiers,
  req_cols = c("cellline", "cellline_name", "cellline_tissue", "cellline_ref_div_time")
)
```

**Arguments**

cdata	data.table of colData.
identifiers	charvec with identifiers
req_cols	charvec required columns

**Details**

Standardizes the cdata to common schema fields and tidies formatting to be conducive to joining with other JSON responses.

**Value**

JSON string capturing the cdata.

**Examples**

```

cdata <- data.table::data.table(
  mycellline = letters,
  mycelllinename = letters,
  mycelllinetissue = letters,
  cellline_ref_div_time = "cellline_ref_div_time")
identifiers <- list(cellline = "mycellline",
                   cellline_name = "mycelllinename",
                   cellline_ref_div_time = "cellline_ref_div_time",
                   cellline_tissue = "mycelllinetissue")
convert_colData_to_json(cdata, identifiers)

```

---

```
convert_combo_data_to_dt
```

*convert combo assays from SummarizedExperiments to the list of data.tables*

---

**Description**

convert combo assays from SummarizedExperiments to the list of data.tables

**Usage**

```

convert_combo_data_to_dt(
  se,
  c_assays = get_combo_assay_names(),
  normalization_type = c("RV", "GR"),
  prettify = TRUE
)

```

**Arguments**

se	SummarizedExperiment object with dose-response data
c_assays	charvec of combo assays to be used
normalization_type	charvec of normalization_types expected in the data
prettify	boolean flag indicating whether or not to prettify the colnames of the returned data

**Value**

list of data.table(s) with combo data

**Author(s)**

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

**Examples**

```
mae <- get_synthetic_data("finalMAE_combo_matrix_small.qs")
convert_combo_data_to_dt(mae[[1]])
```

---

```
convert_combo_field_to_assay
  get combo assay names based on the field name
```

---

**Description**

get combo assay names based on the field name

**Usage**

```
convert_combo_field_to_assay(field)
```

**Arguments**

field                   String containing name of the field for which the assay name should be returned

**Value**

charvec

**Examples**

```
convert_combo_field_to_assay("hsa_score")
```

---

```
convert_mae_assay_to_dt
  Convert a MultiAssayExperiment assay to a long data.table
```

---

**Description**

Convert an assay within a [SummarizedExperiment](#) object in a MultiAssayExperiment to a long data.table.

**Usage**

```
convert_mae_assay_to_dt(
  mae,
  assay_name,
  experiment_name = NULL,
  include_metadata = TRUE,
  retain_nested_rownames = FALSE,
  wide_structure = FALSE
)
```

## Arguments

- mae** A [MultiAssayExperiment](#) object holding experiments with raw and/or processed dose-response data in its assays.
- assay\_name** String of name of the assay to transform within an experiment of the mae.
- experiment\_name** String of name of the experiment in mae whose assay\_name should be converted. Defaults to NULL to indicate to convert assay in all experiments into one data.table object.
- include\_metadata** Boolean indicating whether or not to include rowData() and colData() in the returned data.table. Defaults to TRUE.
- retain\_nested\_rownames** Boolean indicating whether or not to retain the rownames nested within a BumpyMatrix assay. Defaults to FALSE. If the assay\_name is not of the BumpyMatrix class, this argument's value is ignored. If TRUE, the resulting column in the data.table will be named as "<assay\_name>\_rownames".
- wide\_structure** Boolean indicating whether or not to transform data.table into wide format. wide\_structure = TRUE requires retain\_nested\_rownames = TRUE however that will be validated in convert\_se\_assay\_to\_dt function

## Details

NOTE: to extract information about 'Control' data, simply call the function with the name of the assay holding data on controls.

## Value

data.table representation of the data in assay\_name.

## Author(s)

Bartosz Czech [bartosz.czech@contractors.roche.com](mailto:bartosz.czech@contractors.roche.com)

## See Also

flatten convert\_se\_assay\_to\_dt

## Examples

```
mae <- get_synthetic_data("finalMAE_small")
convert_mae_assay_to_dt(mae, "Metrics")
```

---

convert\_mae\_to\_json     *Create JSON document.*

---

**Description**

Convert a MultiAssayExperiment object to a JSON document.

**Usage**

```
convert_mae_to_json(mae, with_experiments = TRUE)
```

**Arguments**

mae                     SummarizedExperiment object.  
with\_experiments       logical convert experiment metadata as well?

**Value**

String representation of a JSON document.

**Examples**

```
mae <- get_synthetic_data("finalMAE_small")  
convert_mae_to_json(mae)  
convert_mae_to_json(mae, with_experiments = FALSE)
```

---

convert\_metadata\_to\_json  
                          *Convert experiment metadata to JSON*

---

**Description**

Convert experiment metadata to JSON format for elasticsearch indexing.

**Usage**

```
convert_metadata_to_json(se)
```

**Arguments**

se                         SummarizedExperiment object.

**Value**

JSON string capturing experiment metadata.

## Examples

```
md <- list(title = "my awesome experiment",
  description = "description of experiment",
  sources = list(list(name = "GeneData_Screener", id = "QCS-12345")))
se <- SummarizedExperiment::SummarizedExperiment(metadata = md)
convert_metadata_to_json(se)
```

---

convert\_rowData\_to\_json

*Convert rowData to JSON*

---

## Description

Convert rowData to JSON format for elasticsearch indexing.

## Usage

```
convert_rowData_to_json(
  rdata,
  identifiers,
  req_cols = c("drug", "drug_name", "drug_moa", "duration")
)
```

## Arguments

rdata	data.table of rowData.
identifiers	charvec with identifiers
req_cols	charvec required columns

## Details

Standardizes the rdata to common schema fields and tidies formatting to be conducive to joining with other JSON responses.

## Value

JSON string capturing the rdata.

## Examples

```
rdata <- data.table::data.table(
  mydrug = letters,
  mydrugname = letters,
  mydrugmoa = letters,
  Duration = 1)
identifiers <- list(drug = "mydrug", drug_name = "mydrugname", drug_moa = "mydrugmoa",
  duration = "Duration")
```



```
convert_rowData_to_json(rdata, identifiers)
```

---

```
convert_se_assay_to_dt
```

*Convert a SummarizedExperiment assay to a long data.table*

---

## Description

Convert an assay within a [SummarizedExperiment](#) object to a long data.table.

## Usage

```
convert_se_assay_to_dt(
  se,
  assay_name,
  include_metadata = TRUE,
  retain_nested_rownames = FALSE,
  wide_structure = FALSE
)
```

## Arguments

se	A <a href="#">SummarizedExperiment</a> object holding raw and/or processed dose-response data in its assays.
assay_name	String of name of the assay to transform within the se.
include_metadata	Boolean indicating whether or not to include rowData(se) and colData(se) in the returned data.table. Defaults to TRUE.
retain_nested_rownames	Boolean indicating whether or not to retain the rownames nested within a BumpyMatrix assay. Defaults to FALSE. If the assay_name is not of the BumpyMatrix class, this argument's value is ignored. If TRUE, the resulting column in the data.table will be named as "<assay_name>_rownames".
wide_structure	Boolean indicating whether or not to transform data.table into wide format. wide_structure = TRUE requires retain_nested_rownames = TRUE.

## Details

NOTE: to extract information about 'Control' data, simply call the function with the name of the assay holding data on controls. To extract the reference data in to same format as 'Averaged' use convert\_se\_ref\_assay\_to\_dt.

## Value

data.table representation of the data in assay\_name.

**See Also**

flatten

**Examples**

```
mae <- get_synthetic_data("finalMAE_small")
se <- mae[[1]]
convert_se_assay_to_dt(se, "Metrics")
```

---

convert\_se\_to\_json      *Create JSON document.*

---

**Description**

Convert a SummarizedExperiment object to a JSON document.

**Usage**

```
convert_se_to_json(se)
```

**Arguments**

se                      SummarizedExperiment object.

**Value**

String representation of a JSON document.

**Examples**

```
md <- list(title = "my awesome experiment",
  description = "description of experiment",
  source = list(name = "GeneData_Screener", id = "QCS-12345"))
rdata <- data.table::data.table(
  mydrug = letters,
  mydrugname = letters,
  mydrugmoa = letters,
  Duration = 1)
cdata <- data.table::data.table(mycellline = letters, mycelllinename = letters,
  mycelllinetissue = letters, cellline_ref_div_time = letters)
identifiers <- list(cellline = "mycellline",
  cellline_name = "mycelllinename",
  cellline_tissue = "mycelllinetissue",
  cellline_ref_div_time = "cellline_ref_div_time",
  drug = "mydrug",
  drug_name = "mydrugname",
  drug_moa = "mydrugmoa",
  duration = "Duration")
```

```
se <- SummarizedExperiment::SummarizedExperiment(rowData = rdata,
                                                  colData = cdata)
se <- set_SE_experiment_metadata(se, md)
se <- set_SE_identifiers(se, identifiers)
convert_se_to_json(se)
```

---

demote_fields	<i>Demote a metadata field in the rowData or colData of a SummarizedExperiment object to a nested field of a BumpyMatrix assay.</i>
---------------	---

---

### Description

Demote a metadata field in the `rowData` or `colData` of a `SummarizedExperiment` object to a nested field of a `BumpyMatrix` assay.

### Usage

```
demote_fields(se, fields)
```

### Arguments

<code>se</code>	A <code>SummarizedExperiment</code> object.
<code>fields</code>	Character vector of metadata fields to demote as nested columns.

### Details

Revert this operation using `promote_fields`.

### Value

A `SummarizedExperiment` object with new dimensions resulting from demoting given fields to nested columns.

### See Also

`promote_fields`

### Examples

```
mae <- get_synthetic_data("finalMAE_small")
se <- mae[[1]]
se <- promote_fields(se, "ReadoutValue", 2)
demote_fields(se, "ReadoutValue")
```

---

df_to_bm_assay	<i>df_to_bm_assay</i>
----------------	-----------------------

---

**Description**

Convert data.table with dose-response data into a BumpyMatrix assay.

**Usage**

```
df_to_bm_assay(data, discard_keys = NULL)
```

**Arguments**

data	data.table with drug-response data
discard_keys	a vector of keys that should be discarded

**Details**

The 'assay' is simply a BumpyMatrix object with rownames being the treatment ids, colnames being the ids of the cell lines and values with dose-response data for given cell lines under given conditions.

**Value**

BumpyMatrix object

**Examples**

```
df_to_bm_assay(data.table::data.table(Gnumber = 2, clid = "A"))
```

---

extend_normalization_type_name	<i>extend abbreviated normalization type</i>
--------------------------------	--

---

**Description**

extend abbreviated normalization type

**Usage**

```
extend_normalization_type_name(x)
```

**Arguments**

x	string with normalization type
---	--------------------------------

**Value**

string

**Examples**

```
extend_normalization_type_name("GR")
```

---

fit\_curves

*Fit curves*


---

**Description**

Fit GR and RV curves from a data.table.

**Usage**

```
fit_curves(
  df_,
  series_identifiers,
  e_0 = 1,
  GR_0 = 1,
  n_point_cutoff = 4,
  range_conc = c(0.005, 5),
  force_fit = FALSE,
  pcutoff = 0.05,
  cap = 0.1,
  normalization_type = c("GR", "RV")
)
```

**Arguments**

df_	data.table containing data to fit. See details.
series_identifiers	character vector of the column names in data.table whose combination represents a unique series for which to fit curves.
e_0	numeric value representing the x_0 value for the RV curve. Defaults to 1.
GR_0	numeric value representing the x_0 value for the GR curve. Defaults to 1.
n_point_cutoff	integer of how many points should be considered the minimum required to try to fit a curve. Defaults to 4.
range_conc	numeric vector of length 2 indicating the lower and upper concentration ranges. Defaults to c(5e-3, 5). See details.
force_fit	boolean indicating whether or not to force a constant fit. Defaults to FALSE.
pcutoff	numeric of pvalue significance threshold above or equal to which to use a constant fit. Defaults to 0.05.

cap numeric value capping norm\_values to stay below  $(x_0 + \text{cap})$ . Defaults to 0.1.  
 normalization\_type character vector of types of curves to fit. Defaults to c("GR", "RV").

### Details

The df\_ expects the following columns:

- RelativeViability normalized relative viability values (if normalization\_type includes "RV")
- GRvalue normalized GR values (if normalization\_type includes "GR")

The range\_conc is used to calculate the x\_AOC\_range statistic. The purpose of this statistic is to enable comparison across different experiments with slightly different concentration ranges.

### Value

data.table of fit parameters as specified by the normalization\_type.

### Examples

```
df_ <- data.table::data.table(Concentration = c(0.001, 0.00316227766016838,
0.01, 0.0316227766016838),
x_std = c(0.1, 0.1, 0.1, 0.1), normalization_types = c("RV", "RV", "RV", "RV"),
x = c(0.9999964000144, 0.999964001439942, 0.999640143942423, 0.996414342629482))

fit_curves(df_, "Concentration", normalization_type = "RV")
```

---

flatten

*Flatten a table*

---

### Description

Flatten a stacked table into a wide format.

### Usage

```
flatten(tbl, groups, wide_cols, sep = "_")
```

### Arguments

tbl table to flatten.  
 groups character vector of column names representing unifying groups in expansion.  
 wide\_cols character vector of column names to flatten.  
 sep string representing separator between wide\_cols columns, used in column re-naming. Defaults to "\_".

**Details**

flattened columns will be named with original column names prefixed by wide\_cols columns, concatenated together and separated by sep.

A common use case for this function is when a flattened version of the "Metrics" assay is desired.

**Value**

table of flattened data as defined by wide\_cols.

**See Also**

convert\_se\_assay\_to\_dt

**Examples**

```
n <- 4
m <- 5
grid <- expand.grid(normalization_type = c("GR", "RV"),
  source = c("GDS", "GDR"))
regrid <- data.table::rbindlist(rep(list(grid), m))
regrid$wide <- seq(m * n)
regrid$id <- rep(LETTERS[1:m], each = n)

groups <- colnames(grid)
wide_cols <- c("wide")

flatten(regrid, groups = groups, wide_cols = wide_cols)
```

---

gen\_synthetic\_data      *gen\_synthetic\_data*

---

**Description**

Function for generating local synthetic data used for unit tests in modules

**Usage**

```
gen_synthetic_data(m = 1, n = 5)
```

**Arguments**

m	number of drugs
n	number of records

**Value**

list with drugs, cell\_lines, raw\_data and assay\_data

**Examples**

```
gen_synthetic_data()
```

---

geometric_mean	<i>Geometric mean</i>
----------------	-----------------------

---

**Description**

Auxiliary function for calculating geometric mean with possibility to handle -Inf

**Usage**

```
geometric_mean(x, fixed = TRUE, maxlog10Concentration = 1)
```

**Arguments**

x	numeric vector
fixed	flag should be add fix for -Inf
maxlog10Concentration	numeric value needed to calculate minimal value

**Value**

numeric vector

**Examples**

```
geometric_mean(c(2, 8))
```

---

get_assay_names	<i>get assay names of the given se/dataset fetch the data from the se if provided as metadata use predefined values from get_env_assay_names otherwise</i>
-----------------	--

---

**Description**

get assay names of the given se/dataset fetch the data from the se if provided as metadata use predefined values from get\_env\_assay\_names otherwise

**Usage**

```
get_assay_names(se = NULL, ...)
```



**Arguments**

`se` SummarizedExperiment or NULL  
`...` Additional arguments to pass to `get_env_assay_names`.

**Value**

`charvec`

**Author(s)**

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

**Examples**

```
get_assay_names()
```

---

`get_combo_assay_names` *get names of combo assays*

---

**Description**

get names of combo assays

**Usage**

```
get_combo_assay_names(se = NULL, ...)
```

**Arguments**

`se` SummarizedExperiment or NULL  
`...` Additional arguments to pass to `get_assay_names`.

**Value**

`charvec` of combo assay names.

**Author(s)**

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

**Examples**

```
get_combo_assay_names()
```

get\_combo\_base\_assay\_names  
*get names of combo base assays*

---

**Description**

get names of combo base assays

**Usage**

```
get_combo_base_assay_names(se = NULL, ...)
```

**Arguments**

se	SummarizedExperiment or NULL
...	Additional arguments to pass to get_combo_assay_names.

**Value**

charvec

**Author(s)**

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

**Examples**

```
get_combo_base_assay_names()
```

---

get\_combo\_col\_settings  
*Get colorscale data for given combo assay and growth metric*

---

**Description**

Get colorscale data for given combo assay and growth metric

**Usage**

```
get_combo_col_settings(g_metric, assay_type)
```

**Arguments**

g_metric	growth metric
assay_type	assay type

**Value**

list with colors, breaks and limits

**Examples**

```
get_combo_col_settings("GR", "excess")
```

---

```
get_combo_excess_field_names  
get names of combo excess fields
```

---

**Description**

get names of combo excess fields

**Usage**

```
get_combo_excess_field_names()
```

**Value**

charvec

**Examples**

```
get_combo_excess_field_names()
```

---

```
get_combo_score_assay_names  
get names of combo score assays
```

---

**Description**

get names of combo score assays

**Usage**

```
get_combo_score_assay_names(se = NULL, ...)
```

**Arguments**

<code>se</code>	SummarizedExperiment or NULL
<code>...</code>	Additional arguments to pass to <code>get_combo_assay_names</code> .

**Value**

charvec

**Author(s)**

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

**Examples**

```
get_combo_score_assay_names()
```

---

```
get_combo_score_field_names
```

*get names of combo score fields*

---

**Description**

get names of combo score fields

**Usage**

```
get_combo_score_field_names()
```

**Value**

charvec

**Examples**

```
get_combo_score_assay_names()
```

---

```
get_default_identifiers
```

*Get gDR default identifiers required for downstream analysis.*

---

**Description**

Get gDR default identifiers required for downstream analysis.

**Usage**

```
get_default_identifiers()
```

**Value**

charvec

**Examples**

```
get_default_identifiers()
```

---

get\_duplicated\_rows     *Helper function to find duplicated rows*

---

**Description**

Helper function to find duplicated rows

**Usage**

```
get_duplicated_rows(x, col_names = NULL)
```

**Arguments**

x                    data frame  
col\_names            character vector, columns in which duplication are searched for

**Value**

integer vector

**Examples**

```
dt <- data.table::data.table(a = c(1, 2, 3), b = c(3, 2, 2))  
get_duplicated_rows(dt, "b")
```

---

get\_env\_assay\_names     *get default assay names for the specified filters, i.e. set of assay types, assay groups and assay data types*

---

**Description**

get default assay names for the specified filters, i.e. set of assay types, assay groups and assay data types

**Usage**

```
get_env_assay_names(  
  type = NULL,  
  group = NULL,  
  data_type = NULL,  
  prettify = FALSE,  
  simplify = TRUE  
)
```

**Arguments**

type	charvec of assay types
group	charvec of assay groups
data_type	charvec assay of data types
prettify	logical flag, prettify the assay name?
simplify	logical flag, simplify the output? will return single string instead of named vector with single element useful when function is expected to return single element/assay only

**Value**

charvec

**Author(s)**

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

**Examples**

```
get_env_assay_names()
```

---

get\_expect\_one\_identifiers

*Get identifiers that expect only one value for each identifier.*

---

**Description**

Get identifiers that expect only one value for each identifier.

**Usage**

```
get_expect_one_identifiers()
```

**Value**

charvec

**Examples**

```
get_expect_one_identifiers()
```

---

*get\_experiment\_groups*    *get\_experiment\_groups*

---

**Description**

get experiment groups

**Usage**

```
get_experiment_groups(type = NULL)
```

**Arguments**

type                    String indicating the name of an assay group. Defaults to all experiment groups.

**Value**

list with experiment groups or string (if type not NULL)

**Author(s)**

Arkadiusz Gladki [arkadiusz.gladki@contractors.roche.com](mailto:arkadiusz.gladki@contractors.roche.com)

**Examples**

```
get_experiment_groups()
```

---

*get\_identifiers\_dt*    *Get descriptions for identifiers*

---

**Description**

Get descriptions for identifiers

**Usage**

```
get_identifiers_dt(k = NULL, get_description = FALSE, get_example = FALSE)
```

**Arguments**

k identifier key, string  
get\_description return descriptions only, boolean  
get\_example return examples only, boolean

**Value**

named list

**Examples**

```
get_identifiers_dt()
```

---

get\_idfs\_synonyms *Get gDR synonyms for the identifiers*

---

**Description**

Get gDR synonyms for the identifiers

**Usage**

```
get_idfs_synonyms()
```

**Value**

charvec

**Examples**

```
get_idfs_synonyms()
```



---

get\_iso\_colors      *get\_iso\_colors*

---

**Description**

get\_iso\_colors

**Usage**

```
get_iso_colors(normalization_type = c("RV", "GR"))
```

**Arguments**

normalization\_type  
charvec normalization\_types expected in the data

**Value**

named charvec with iso colors

**Examples**

```
get_iso_colors()
```

---

get\_MAE\_identifiers      *get\_MAE\_identifiers*

---

**Description**

get the identifiers of all SE's in the MAE

**Usage**

```
get_MAE_identifiers(mae)
```

**Arguments**

mae      MultiAssayExperiment

**Value**

named list with identifiers for each SE

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")  
get_MAE_identifiers(mae)
```

---

```
get_non_empty_assays  get_non_empty_assays
```

---

**Description**

get non empty assays

**Usage**

```
get_non_empty_assays(mae)
```

**Arguments**

mae                    MultiAssayExperiment object

**Value**

charvec with non-empty experiments

**Author(s)**

Arkadiusz Gladki [arkadiusz.gladki@contractors.roche.com](mailto:arkadiusz.gladki@contractors.roche.com)

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
get_non_empty_assays(mae)
```

---

```
get_optional_coldata_fields
get optional colData fields
```

---

**Description**

get optional colData fields

**Usage**

```
get_optional_coldata_fields(se)
```

**Arguments**

se                    a SummarizedExperiment object with drug-response data generate by gDR pipeline

**Value**

a charvec containing the names of the optional identifiers in the SE colData

---

`get_optional_rowdata_fields`  
*get optional rowData fields*

---

**Description**

get optional rowData fields

**Usage**

`get_optional_rowdata_fields(se)`

**Arguments**

`se` a SummarizedExperiment object with drug-response data generate by gDR pipeline

**Value**

a charvec containing the names of the optional identifiers in the SE rowData

---

`get_required_identifiers`  
*Get identifiers required for downstream analysis.*

---

**Description**

Get identifiers required for downstream analysis.

**Usage**

`get_required_identifiers()`

**Value**

charvec

**Examples**

`get_required_identifiers()`

---

get\_synthetic\_data      *Get synthetic data from gDRtestData package*

---

**Description**

Get synthetic data from gDRtestData package

**Usage**

```
get_synthetic_data(qs)
```

**Arguments**

qs                      qs filename

**Value**

loaded data

**Examples**

```
get_synthetic_data("finalMAE_small.qs")
```

---

get\_testdata              *get\_testdata*

---

**Description**

Function to obtain data from gDRtestData and prepare for unit tests

**Usage**

```
get_testdata()
```

**Value**

list with drugs, cell\_lines, raw\_data and assay\_data

**Examples**

```
get_testdata()
```

---

`get_testdata_codilution`  
*get\_testdata\_codilution*

---

**Description**

Function to obtain data from gDRtestData and prepare for unit tests

**Usage**

`get_testdata_codilution()`

**Value**

list with `drugs`, `cell_lines`, `raw_data` and `assay_data`

**Examples**

`get_testdata_codilution()`

---

`get_testdata_combo`     *get\_testdata\_combo*

---

**Description**

Function to obtain data from gDRtestData and prepare for unit tests

**Usage**

`get_testdata_combo()`

**Value**

list with `drugs`, `cell_lines`, `raw_data` and `assay_data`

**Examples**

`get_testdata_combo()`

---

headers	<i>Get or reset headers for one or all header field(s) respectively</i>
---------	---

---

**Description**

Get the expected header(s) for one field or reset all header fields

**Usage**

```
get_header(k = NULL)
```

**Arguments**

k                    string of field (data type) to return headers for

**Details**

If `get_header` is called with no values, the entire available header list is returned.

**Value**

For `get_header` a character vector of headers for field `k`.

**Examples**

```
get_header(k = NULL)
get_header("manifest")
```

---

identifiers	<i>Get, set, or reset identifiers for one or all identifier field(s)</i>
-------------	--

---

**Description**

Get, set, or reset the expected identifier(s) for one or all identifier field(s). Identifiers are used by the `gDR` processing functions to identify which columns in a `data.table` correspond to certain expected fields. Functions of the family `*et_identifier` will look for identifiers from the environment while functions of the family `*et_SE_identifiers` will look for identifiers in the metadata slot of a `SummarizedExperiment` object. See details for expected identifiers and their definitions.

**Usage**

```
get_env_identifiers(k = NULL, simplify = TRUE)
get_prettified_identifiers(k = NULL, simplify = TRUE)
set_env_identifier(k, v)
reset_env_identifiers()
```

**Arguments**

k	String corresponding to identifier name.
simplify	Boolean indicating whether output should be simplified.
v	Character vector corresponding to the value for given identifier k.

**Details**

Identifiers supported by the gDR suite include:

- "barcode": String of column name containing barcode metadata
- "cellline": String of column name containing unique, machine-readable cell line identifiers
- "cellline\_name": String of column name containing human-friendly cell line names
- "cellline\_tissue": String of column name containing metadata on cell line tissue type
- "cellline\_ref\_div\_time": String of column name containing reference division time for cell lines
- "cellline\_parental\_identifier": String of column name containing unique, machine-readable parental cell line identifiers. Used in the case of derived or engineered cell lines.
- "drug": String of column name containing unique, machine-readable drug identifiers
- "drug\_name": String of column name containing human-friendly drug names
- "drug\_moa": String of column name containing metadata for drug mode of action
- "duration": String of column name containing metadata on duration that cells were treated (in hours)
- "template": String of column name containing template metadata
- "untreated\_tag": Character vector of entries that identify control, untreated wells
- "well\_position": Character vector of column names containing metadata on well positions on a plate

**Value**

For any setting or resetting functionality, a NULL invisibly. For `get_env_identifiers` a character vector of identifiers for field k. For functions called with no arguments, the entire available identifier list is returned.

list or charvec depends on unify param

list or charvec depends on unify param

NULL

NULL

**Examples**

```
get_env_identifiers("duration") # "Duration"
```

---

```
identify_unique_se_metadata_fields
```

*Identify unique metadata fields from a list of SummarizedExperiments*

---

**Description**

Identify unique metadata fields from a list of SummarizedExperiments

**Usage**

```
identify_unique_se_metadata_fields(SElist)
```

**Arguments**

SElist            named list of SummarizedExperiments

**Value**

character vector of unique names of metadata

**Examples**

```
mae <- get_synthetic_data("finalMAE_small")
se <- mae[[1]]
SElist <- list(
  se,
  se
)
identify_unique_se_metadata_fields(SElist)
```

---

```
is_any_exp_empty        is_any_exp_empty
```

---

**Description**

check if any experiment is empty

**Usage**

```
is_any_exp_empty(mae)
```

**Arguments**

mae                MultiAssayExperiment object



**Value**

logical

**Author(s)**

Arkadiusz Gladki [arkadiusz.gladki@contractors.roche.com](mailto:arkadiusz.gladki@contractors.roche.com)

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
is_any_exp_empty(mae)
```

---

<i>is_exp_empty</i>	<i>is_exp_empty</i>
---------------------	---------------------

---

**Description**

check if experiment (SE) is empty

**Usage**

```
is_exp_empty(exp)
```

**Arguments**

exp                    [SummarizedExperiment](#) object.

**Value**

logical

**Author(s)**

Arkadiusz Gladki [arkadiusz.gladki@contractors.roche.com](mailto:arkadiusz.gladki@contractors.roche.com)

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
is_exp_empty(se)
```

---

is_mae_empty	<i>is_mae_empty</i>
--------------	---------------------

---

**Description**

check if all mae experiments are empty

**Usage**

```
is_mae_empty(mae)
```

**Arguments**

mae                    MultiAssayExperiment object

**Value**

logical

**Author(s)**

Arkadiusz Gladki [arkadiusz.gladki@contractors.roche.com](mailto:arkadiusz.gladki@contractors.roche.com)

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
is_mae_empty(mae)
```

---

logisticFit	<i>Logistic fit</i>
-------------	---------------------

---

**Description**

Fit a logistic curve to drug response data.

**Usage**

```
logisticFit(
  concs,
  norm_values,
  std_norm_values = NA,
  x_0 = 1,
  priors = NULL,
  lower = NULL,
  range_conc = c(0.005, 5),
```

```

    force_fit = FALSE,
    pcutoff = 0.05,
    cap = 0.1,
    n_point_cutoff = 4,
    capping_fold = 5
)

```

### Arguments

concs	concentrations that have not been transformed into log space.
norm_values	normalized response values (Untreated = 1).
std_norm_values	std of values.
x_0	upper limit. Defaults to 1. For co-treatments, this value should be set to NA.
priors	numeric vector containing starting values for all. mean parameters in the model. Overrides any self starter function.
lower	numeric vector of lower limits for all parameters in a 4-param model.
range_conc	range of concentration for calculating AOC_range.
force_fit	boolean indicating whether or not to force a parameter-based fit.
pcutoff	numeric of pvalue significance threshold above or equal to which to use a constant fit.
cap	numeric value capping norm_values to stay below $(x_0 + cap)$ .
n_point_cutoff	integer indicating number of unique concentrations required to fit curve.
capping_fold	Integer value of the fold number to use for capping IC50/GR50. Default is 5.

### Details

Implementation of the genedata approach for curve fit: [#nolint](https://screener.genedata.com/documentation/display/DOC15/Business+Response+Curve+Fitting+Model+Selection+and+Fit+Validity)

The output parameter names correspond to the following definitions:

**x\_mean** The mean of a given dose-response metric

**x\_AOC\_range** The range of the area over the curve

**x\_AOC** The area over the GR curve or, respectively, under the relative cell count curve, averaged over the range of concentration values

**xc50** The concentration at which the effect reaches a value of 0.5 based on interpolation of the fitted curve

**x\_max** The maximum effect of the drug

**ec50** The drug concentration at half-maximal effect

**x\_inf** The asymptotic value of the sigmoidal fit to the dose-response data as concentration goes to infinity

**x\_0** The asymptotic metric value corresponding to a concentration of 0 for the primary drug

**h** The hill coefficient of the fitted curve, which reflects how steep the dose-response curve is

**r2** The goodness of the fit

**x\_sd\_avg** The standard deviation of GR/IC

**fit\_type** This will be given by one of the following:

- "DRC4pHillFitModel" Successfully fit with a 4-parameter model
- "DRC3pHillFitModelFixS0" Successfully fit with a 3-parameter model
- "DRCConstantFitResult" Successfully fit with a constant fit
- "DRCTooFewPointsToFit" Not enough points to run a fit
- "DRCInvalidFitResult" Fit was attempted but failed

**maxlog10Concentration** The highest log10 concentration

**N\_conc** Number of unique concentrations

### Value

data.table with metrics and fit parameters.

### Examples

```
logisticFit(
  c(0.001, 0.00316227766016838, 0.01, 0.0316227766016838),
  c(0.9999964000144, 0.999964001439942, 0.999640143942423, 0.996414342629482),
  rep(0.1, 4),
  priors = c(2, 0.4, 1, 0.00658113883008419)
)
```

---

loop

*Lapply or bplapply.*

---

### Description

Lapply or bplapply.

### Usage

```
loop(x, FUN, parallelize = TRUE, ...)
```

### Arguments

x	Vector (atomic or list) or an ‘expression’ object. Other objects (including classed objects) will be coerced by ‘base::as.list’.
FUN	A user-defined function.
parallelize	Logical indicating whether or not to parallelize the computation. Defaults to TRUE.
...	optional argument passed to <a href="#">bplapply</a> if parallelize == TRUE, else to <a href="#">lapply</a> .

**Value**

List containing output of FUN applied to every element in x.

**Examples**

```
loop(list(c(1,2), c(2,3)), sum, parallelize = FALSE)
```

---

MAEapply

*Lapply through all the experiments in MultiAssayExperiment object*

---

**Description**

Lapply through all the experiments in MultiAssayExperiment object

**Usage**

```
MAEapply(mae, FUN, unify = FALSE, ...)
```

**Arguments**

mae	MultiAssayExperiment object
FUN	function that should be applied on each experiment of MultiAssayExperiment object
unify	logical indicating if the output should be a unlisted object of unique values across all the experiments
...	Additional args to be passed to teh FUN.

**Value**

list or vector depends on unify param

**Author(s)**

Bartosz Czech [bartosz.czech@contractors.roche.com](mailto:bartosz.czech@contractors.roche.com)

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
MAEapply(mae, SummarizedExperiment::assayNames)
```

---

mcolData	<i>mcolData</i>
----------	-----------------

---

**Description**

get colData of all experiments

**Usage**

```
mcolData(mae)
```

**Arguments**

mae                    MultiAssayExperiment object

**Value**

data.table with all-experiments colData

**Author(s)**

Arkadiusz Gladki [arkadiusz.gladki@contractors.roche.com](mailto:arkadiusz.gladki@contractors.roche.com)

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
mcolData(mae)
```

---

merge_assay	<i>Merge assay data</i>
-------------	-------------------------

---

**Description**

Merge assay data

**Usage**

```
merge_assay(  
  SElist,  
  assay_name,  
  additional_col_name = "data_source",  
  discard_keys = NULL  
)
```

**Arguments**

SElist	named list of Summarized Experiments
assay_name	name of the assay that should be extracted and merged
additional_col_name	string of column name that will be added to assay data for the distinction of possible duplicated metrics that can arise from multiple projects
discard_keys	character vector of string that will be discarded during creating BumpyMatrix object

**Value**

BumpyMatrix or list with data.table + BumpyMatrix

**Examples**

```
mae <- get_synthetic_data("finalMAE_combo_2dose_nonoise")
listSE <- list(
  combo1 = mae[[1]],
  sa = mae[[2]]
)
merge_assay(listSE, "Normalized")
```

---

merge_metadata	<i>Merge metadata</i>
----------------	-----------------------

---

**Description**

Merge metadata

**Usage**

```
merge_metadata(SElist, metadata_fields)
```

**Arguments**

SElist	named list of SummarizedExperiments
metadata_fields	vector of metadata names that will be merged

**Value**

list of merged metadata

## Examples

```
mae <- get_synthetic_data("finalMAE_small")
se <- mae[[1]]
listSE <- list(
  se,
  se
)
metadata_fields <- identify_unique_se_metadata_fields(listSE)
merge_metadata(listSE, metadata_fields)
```

---

merge\_SE

*Merge multiple Summarized Experiments*

---

## Description

Merge multiple Summarized Experiments

## Usage

```
merge_SE(
  SElist,
  additional_col_name = "data_source",
  discard_keys = c("normalization_type", "fit_source", "record_id", "swap_sa",
    "control_type")
)
```

## Arguments

SElist            named list of Summarized Experiments

additional\_col\_name        string with the name of the column that will be added to assay data for the distinction of possible duplicated metrics that can arise from multiple projects

discard\_keys        character vector of string that will be discarded during creating BumpyMatrix object

## Value

merged SummarizedExperiment object

## Examples

```
se1 <- get_synthetic_data("finalMAE_small")[[1]]
merge_SE(list(se1 = se1, se2 = se1))
```



---

modifyData	<i>modify assay with additional data</i>
------------	--

---

## Description

Reduce dimensionality of an assay by dropping extra data or combining variables.

## Usage

```
modifyData(x, ...)  
  
## S3 method for class 'drug_name2'  
modifyData(x, option, keep, ...)  
  
## S3 method for class 'data_source'  
modifyData(x, option, keep, ...)  
  
## Default S3 method:  
modifyData(x, option, keep, ...)
```

## Arguments

x	a <code>data.table</code> containing an assay
...	additional arguments passed to methods
option	character string specifying the action to be taken, see Details
keep	character string specifying the value of the active variable that will be kept

## Details

If an assay extracted from a `SummarizedExperiment` contains additional information, i.e. factors beyond `DrugName` and `CellLineName`, that information will be treated in one of three ways, depending on the value of `option`:

- `drop`: Some information will be discarded and only one value of the additional variable (chosen by the user) will be kept.
- `toDrug`: The information is pasted together with the primary drug name. All observations are kept.
- `toCellLine`: As above, but pasting is done with cell line name.

Depending on the type of additional information, the exact details will differ. This is handled in the app by adding special classes to the data tables and dispatching to S3 methods.

Following modification, the additional columns are discarded.

**Methods (by class)**

- `modifyData(drug_name2)`: includes the name and concentration of the second drug
- `modifyData(data_source)`: includes the data source
- `modifyData(default)`: includes the name of other additional variables

**Examples**

```
dt <- data.table::data.table(a = as.character(1:10), b = "data")
dt <- addClass(dt, "a")
modifyData(dt, "average", keep = "b")
```

---

`mrowData`*mrowData*

---

**Description**

get rowData of all experiments

**Usage**

```
mrowData(mae)
```

**Arguments**

`mae` MultiAssayExperiment object

**Value**

data.table with all-experiments rowData

**Author(s)**

Arkadiusz Gladki [arkadiusz.gladki@contractors.roche.com](mailto:arkadiusz.gladki@contractors.roche.com)

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
mrowData(mae)
```

---

`predict_conc_from_efficacy`*Predict a concentration for a given efficacy with fit parameters.*

---

**Description**

Predict a concentration for a given efficacy with fit parameters.

**Usage**

```
predict_conc_from_efficacy(efficacy, x_inf, x_0, ec50, h)
```

**Arguments**

<code>efficacy</code>	Numeric vector representing efficacies to predict concentrations for.
<code>x_inf</code>	Numeric vector representing the asymptotic value of the sigmoidal fit to the dose-response data as concentration goes to infinity.
<code>x_0</code>	Numeric vector representing the asymptotic metric value corresponding to a concentration of 0 for the primary drug.
<code>ec50</code>	Numeric vector representing the drug concentration at half-maximal effect.
<code>h</code>	Numeric vector representing the hill coefficient of the fitted curve, which reflects how steep

**Details**

The inverse of this function is `predict_efficacy_from_conc`.

**Value**

Numeric vector representing predicted concentrations from given efficacies and fit parameters.

**See Also**

`predict_efficacy_from_conc` `.calculate_x50`

**Examples**

```
predict_conc_from_efficacy(efficacy = c(1, 1.5), x_inf = 0.1, x_0 = 1, ec50 = 0.5, h = 2)
```

---

predict\_efficity\_from\_conc

*Predict efficacy values given fit parameters and a concentration.*

---

### Description

Predict efficacy values given fit parameters and a concentration.

### Usage

```
predict_efficity_from_conc(c, x_inf, x_0, ec50, h)
```

### Arguments

c	Numeric vector representing concentrations to predict efficacies for.
x_inf	Numeric vector representing the asymptotic value of the sigmoidal fit to the dose-response data as concentration goes to infinity.
x_0	Numeric vector representing the asymptotic metric value corresponding to a concentration of 0 for the primary drug.
ec50	Numeric vector representing the drug concentration at half-maximal effect.
h	Numeric vector representing the hill coefficient of the fitted curve, which reflects how steep the dose-response curve is.

### Details

The inverse of this function is `predict_conc_from_efficity`.

### Value

Numeric vector representing predicted efficacies from given concentrations and fit parameters.

### See Also

`predict_conc_from_efficity`

### Examples

```
predict_efficity_from_conc(c = 1, x_inf = 0.1, x_0 = 1, ec50 = 0.5, h = 2)
```

---

prettyfy\_flat\_metrics *Prettify metric names in flat 'Metrics' assay*

---

## Description

Map existing column names of a flattened 'Metrics' assay to prettified names.

## Usage

```
prettyfy_flat_metrics(  
  x,  
  human_readable = FALSE,  
  normalization_type = c("GR", "RV")  
)
```

## Arguments

**x** character vector of names to prettify.

**human\_readable** boolean indicating whether or not to return column names in human readable format. Defaults to FALSE.

**normalization\_type** character vector with a specified normalization type. Defaults to c("GR", "RV").

## Details

A common use case for this function is to prettify column names from a flattened version of the "Metrics" assay. Mode "human\_readable" = TRUE is often used for prettification in the context of front-end applications, whereas "human\_readable" = FALSE is often used for prettification in the context of the command line.

## Value

character vector of prettified names.

## Examples

```
x <- c("CellLineName", "Tissue", "Primary Tissue", "GR_gDR_x_mean", "GR_gDR_xc50", "RV_GDS_x_mean")  
prettyfy_flat_metrics(x, human_readable = FALSE)
```

---

promote_fields	<i>Promote a nested field to be represented as a metadata field of the SummarizedExperiment as either the rowData or colData.</i>
----------------	---

---

### Description

Promote a nested field to be represented as a metadata field of the SummarizedExperiment as either the rowData or colData.

### Usage

```
promote_fields(se, fields, MARGIN = c(1, 2))
```

### Arguments

se	SummarizedExperiment object.
fields	Character vector of nested fields in a BumpyMatrix object to promote to metadata fields of a se.
MARGIN	Numeric of values 1 or 2 indicating whether to promote fields to rows or columns respectively.

### Details

Revert this operation using demote\_fields.

### Value

A SummarizedExperiment object with new dimensions resulting from promoting given fields.

### See Also

demote\_fields

### Examples

```
mae <- get_synthetic_data("finalMAE_small")
se <- mae[[1]]
se <- promote_fields(se, "ReadoutValue", 2)
```

---

refine_coldata	<i>refine colData</i>
----------------	-----------------------

---

### Description

current improvements done on the colData as a standardization step:

- set default value for optional colData fields

### Usage

```
refine_coldata(cd, se, default_v = "Undefined")
```

### Arguments

cd	DataFrame with colData
se	a SummarizedExperiment object with drug-response data generate by gDR pipeline
default_v	string with default value for optional columns in colData

### Value

refined colData

### Examples

```
mae <- get_synthetic_data("finalMAE_small.qs")
refine_coldata(SummarizedExperiment::colData(mae[[1]]), mae[[1]])
```

---

refine_rowdata	<i>refine rowData</i>
----------------	-----------------------

---

### Description

current improvements done on the rowData as a standardization step:

- set default value for optional rowData fields

### Usage

```
refine_rowdata(rd, se, default_v = "Undefined")
```

### Arguments

rd	DataFrame with rowData
se	a SummarizedExperiment object with drug-response data generate by gDR pipeline
default_v	string with default value for optional columns in rowData

**Value**

refined rowData

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
refine_rowdata(SummarizedExperiment::colData(mae[[1]]), mae[[1]])
```

---

rename\_bumpy

*Rename BumpyMatrix*

---

**Description**

Rename BumpyMatrix

**Usage**

```
rename_bumpy(bumpy, mapping_vector)
```

**Arguments**

bumpy            a BumpyMatrix object

mapping\_vector   a named vector for mapping old and new values. The names of the character vector indicate the source names, and the corresponding values the destination names.

**Value**

a renamed BumpyMatrix object

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
assay <- SummarizedExperiment::assay(se)
rename_bumpy(assay, c("Concentration" = "conc"))
```



---

rename_DFrame	<i>Rename DFrame</i>
---------------	----------------------

---

**Description**

Rename DFrame

**Usage**

```
rename_DFrame(df, mapping_vector)
```

**Arguments**

df	a DFrame object
mapping_vector	a named vector for mapping old and new values. The names of the character vector indicate the source names, and the corresponding values the destination names.

**Value**

a renamed DFrame object

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
rename_DFrame(SummarizedExperiment::rowData(mae[[1]]), c("Gnumber" = "Gnumber1"))
```

---

set_constant_fit_params
-------------------------

*Set fit parameters for a constant fit.*

---

**Description**

Replace values for flat fits: ec50 = 0, h = 0.0001 and xc50 = +/- Inf

**Usage**

```
set_constant_fit_params(out, mean_norm_value)
```

**Arguments**

out	Named list of fit parameters.
mean_norm_value	Numeric value that be used to set all parameters that can be calculated from the mean.

**Value**

Modified named list of fit parameters.

**Examples**

```
na <- list(x_0 = NA)
set_constant_fit_params(na, mean_norm_value = 0.6)
```

---

SE_metadata	<i>Get and set metadata for parameters on a SummarizedExperiment object.</i>
-------------	--

---

**Description**

Set metadata for the fitting parameters that define the Metrics assay in SummarizedExperiment object metadata.

**Usage**

```
set_SE_fit_parameters(se, value)
set_SE_processing_metadata(se, value)
set_SE_keys(se, value)
set_SE_experiment_metadata(se, value)
set_SE_experiment_raw_data(se, value)
get_SE_fit_parameters(se)
get_SE_processing_metadata(se)
get_SE_experiment_raw_data(se)
get_SE_experiment_metadata(se)
get_SE_keys(se, key_type = NULL)
get_SE_identifiers(se, id_type = NULL, simplify = TRUE)
set_SE_identifiers(se, value)
```

**Arguments**

se	a <a href="#">SummarizedExperiment</a> object for which to add fit parameter metadata.
value	named list of metadata for fit parameters.
key_type	string of a specific key type (i.e. 'nested_keys', etc.).
id_type	string of a specific id type (i.e. 'duration', 'cellline_name', etc.).
simplify	Boolean indicating whether output should be simplified.

**Details**

For `*et_SE_processing_metadata`, get/set metadata for the processing info that defines the `date_processed` and packages versions in `SummarizedExperiment` object metadata. For `*et_SE_fit_parameters`, get/set metadata for fit parameters used to construct the `Metrics` assay in a `SummarizedExperiment` object.

**Value**

se with added metadata.

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
get_SE_fit_parameters(se)
```

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
meta <- get_SE_processing_metadata(se)
```

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
get_SE_experiment_raw_data(se)
```

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
get_SE_experiment_metadata(se)
```

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
get_SE_identifiers(se)
```

---

shorten\_normalization\_type\_name  
*shorten normalization type*

---

**Description**

shorten normalization type

**Usage**

```
shorten_normalization_type_name(x)
```

**Arguments**

x                      string with normalization type

**Value**

shortened string representing the normalization type

**Examples**

```
shorten_normalization_type_name("GRvalue")
```

---

```
split_SE_components    split_SE_components
```

---

**Description**

Divide the columns of an input `data.table` into treatment metadata, condition metadata, experiment metadata, and assay data for further analysis. This will most commonly be used to identify the different components of a [SummarizedExperiment](#) object.

**Usage**

```
split_SE_components(df_, nested_keys = NULL, combine_on = 1L)
```

**Arguments**

df\_                    `data.table` with drug-response data

nested\_keys          character vector of keys to exclude from the row or column metadata, and to instead nest within an element of the matrix. See details.

combine\_on          integer value of 1 or 2, indicating whether unrecognized columns should be combined on row or column respectively. Defaults to 1.

**Details**

Named list containing the following elements:

- "treatment\_md": treatment metadata
- "condition\_md": condition metadata
- "data\_fields": all `data.table` column names corresponding to fields nested within a `BumpyMatrix` cell
- "experiment\_md": metadata that is constant for all entries of the `data.table`

- "identifiers\_md": key identifier mappings

The `nested_keys` provides the user the opportunity to specify that they would not like to use that metadata field as a differentiator of the treatments, and instead, incorporate it into a nested DataFrame in the BumpyMatrix matrix object.

In the event that if any of the `nested_keys` are constant throughout the whole `data.table`, they will still be included in the DataFrame of the BumpyMatrix and not in the `experiment_metadata`.

Columns within the `df_` will be identified through the following logic: First, the known data fields and any specified `nested_keys` are extracted. Following that, known cell and drug metadata fields are detected, and any remaining columns that represent constant metadata fields across all rows are extracted. Next, any cell line metadata will be heuristically extracted. Finally, all remaining columns will be combined on either the rows or columns as specified by `combine_on`.

### Value

named list containing different elements of a [SummarizedExperiment](#); see details.

### Examples

```
split_SE_components(data.table::data.table(cldid = "CL1", Gnumber = "DrugA"))
```

---

standardize_mae	<i>Standardize MAE by switching from custom identifiers into gDR-default</i>
-----------------	--

---

### Description

Standardize MAE by switching from custom identifiers into gDR-default

### Usage

```
standardize_mae(mae, use_default = TRUE)
```

### Arguments

mae	a MultiAssayExperiment object with drug-response data generate by gDR pipeline
use_default	boolean indicating whether or not to use default identifiers for standardization

### Value

mae a MultiAssayExperiment with default gDR identifiers

### Examples

```
mae <- get_synthetic_data("finalMAE_small.qs")
S4Vectors::metadata(mae[[1]])$identifiers$drug <- "drug"
standardize_mae(mae)
```

---

standardize_se	<i>Standardize SE by switching from custom identifiers into gDR-default</i>
----------------	---

---

**Description**

Standardize SE by switching from custom identifiers into gDR-default

**Usage**

```
standardize_se(se, use_default = TRUE)
```

**Arguments**

se	a SummarizedExperiment object with drug-response data generate by gDR pipeline
use_default	boolean indicating whether or not to use default identifiers for standardization

**Value**

se a SummarizedExperiment with default gDR identifiers

**Examples**

```
mae <- get_synthetic_data("finalMAE_small.qs")
se <- mae[[1]]
S4Vectors::metadata(se)$identifiers$drug <- "drug"
standardize_se(se)
```

---

strip_first_and_last_char	
---------------------------	--

*String first and last characters of a string.*

---

**Description**

String first and last characters of a string.

**Usage**

```
strip_first_and_last_char(jstring)
```

**Arguments**

jstring	String of any number of characters greater than 1.
---------	--

**Details**

This is most often used to remove the JSON brackets '{' and '}'.

**Value**

String with first and last characters stripped.

---

```
update_env_idfs_from_mae
```

*Update environment identifiers from MAE object identifiers*

---

**Description**

Update environment identifiers from MAE object identifiers

**Usage**

```
update_env_idfs_from_mae(mae_idfs)
```

**Arguments**

mae\_idfs            A list containing MAE identifiers

**Value**

NULL

**Examples**

```
update_env_idfs_from_mae(list(get_env_identifiers()))
```

---

```
update_idfs_synonyms    Update gDR synonyms for the identifier
```

---

**Description**

Update gDR synonyms for the identifier

**Usage**

```
update_idfs_synonyms(data, dict = get_idfs_synonyms())
```

**Arguments**

data                list of charvec with identifiers data  
dict                list with dictionary

**Value**

list

**Examples**

```
mdict <- list(duration = "time")
iv <- c("Time", "Duration", "time")
update_idfs_synonyms(iv, dict = mdict)
```

---

validate_dimnames	<i>Validate dimnames</i>
-------------------	--------------------------

---

**Description**

Assure that dimnames of two objects are the same

**Usage**

```
validate_dimnames(obj, obj2, skip_empty = TRUE)
```

**Arguments**

obj	first object with dimnames to compare
obj2	second object with dimnames to compare
skip_empty	a logical indicating whether to skip comparison if a given dimname is empty in the case of both objects

**Value**

NULL

---

validate_identifiers	<i>Check that specified identifier values exist in the data.</i>
----------------------	--

---

**Description**

Check that specified identifier values exist in the data and error otherwise.

**Usage**

```
validate_identifiers(
  df,
  identifiers = NULL,
  req_ids = NULL,
  exp_one_ids = NULL
)
```



**Arguments**

df	data.table with colnames.
identifiers	Named list of identifiers where the names are standardized identifier names. If not passed, defaults to <code>get_env_identifiers()</code> .
req_ids	Character vector of standardized identifier names required to pass identifier validation.
exp_one_ids	Character vector of standardized identifiers names where only one identifier value is expected. If not passed, defaults to <code>get_expect_one_identifiers()</code> .

**Details**

Note that this does NOT set the identifiers anywhere (i.e. environment or SummarizedExperiment object). If identifiers do not validate, will throw error as side effect.

**Value**

Named list of identifiers modified to pass validation against the input data. Errors with explanatory message if validation cannot pass with the given identifiers and data.

**Examples**

```
validate_identifiers(
  S4Vectors::DataFrame("Barcode" = NA, "Duration" = NA, "Template" = NA, "clid" = NA),
  req_ids = "barcode"
)
```

---

validate_json	<i>Validate JSON against a schema.</i>
---------------	--

---

**Description**

Validate JSON describing an object against a schema.

**Usage**

```
validate_json(json, schema_path)
```

**Arguments**

json	String of JSON in memory.
schema_path	String of the schema to validate against.

**Details**

This is most often used to validate JSON before passing it in as a document to an ElasticSearch index.

**Value**

Boolean of whether or not JSON successfully validated.

**Examples**

```
json <- '{}'
```

---

validate_MAE	<i>Validate MultiAssayExperiment object</i>
--------------	---

---

**Description**

Function validates correctness of SE included in MAE by checking multiple cases:

- detection of duplicated rowData/colData,
- incompatibility of rownames/colnames,
- occurrence of necessary assays,
- detection of mismatch of CLIDs inside colData and colnames (different order),
- correctness of metadata names.

**Usage**

```
validate_MAE(mae)
```

**Arguments**

mae                    MultiAssayExperiment object produced by the gDR pipeline

**Value**

NULL invisibly if the MultiAssayExperiment is valid. Throws an error if the MultiAssayExperiment is not valid.

**Author(s)**

Bartosz Czech [bartosz.czech@contractors.roche.com](mailto:bartosz.czech@contractors.roche.com)

**Examples**

```
mae <- get_synthetic_data("finalMAE_small")  
validate_MAE(mae)
```

---

`validate_mae_with_schema`*Validate MAE against a schema.*

---

### Description

Validate MAE object against a schema. Currently only SEs are validated TODO: add mae.json schema and validate full MAE object

### Usage

```
validate_mae_with_schema(  
  mae,  
  schema_package = Sys.getenv("SCHEMA_PACKAGE", "gDRutils"),  
  schema_dir_path = Sys.getenv("SCHEMA_DIR_PATH", "schemas"),  
  schema = c(se = "se.json", mae = "mae.json")  
)
```

### Arguments

<code>mae</code>	MultiAssayExperiment object
<code>schema_package</code>	string name of the package with JSON schema files
<code>schema_dir_path</code>	path to the dir with JSON schema files
<code>schema</code>	named charvec with filenames of schemas to validate against.

### Value

Boolean of whether or not mae is valid

### Examples

```
mae <- get_synthetic_data("finalMAE_small")  
validate_mae_with_schema(mae)
```

---

`validate_SE`*Validate SummarizedExperiment object*

---

**Description**

Function validates correctness of SE by checking multiple cases:

- detection of duplicated rowData/colData,
- incompatibility of rownames/colnames,
- occurrence of necessary assays,
- detection of mismatch of CLIDs inside colData and colnames (different order),
- correctness of metadata names.

**Usage**

```
validate_SE(se, expect_single_agent = FALSE)
```

**Arguments**

`se` SummarizedExperiment object produced by the gDR pipeline  
`expect_single_agent` a logical indicating if the function should check whether the SummarizedExperiment is single-agent data

**Value**

NULL invisibly if the SummarizedExperiment is valid. Throws an error if the SummarizedExperiment is not valid.

**Examples**

```
mae <- get_synthetic_data("finalMAE_small")
se <- mae[[1]]
validate_SE(se)
```

---

```
validate_se_assay_name
```

*Check whether or not an assay exists in a SummarizedExperiment object.*

---

**Description**

Check for the presence of an assay in a SummarizedExperiment object.

**Usage**

```
validate_se_assay_name(se, name)
```

**Arguments**

se                    A [SummarizedExperiment](#) object.  
name                 String of name of the assay to validate.

**Value**

NULL invisibly if the assay name is valid. Throws an error if the assay is not valid.

**Examples**

```
mae <- get_synthetic_data("finalMAE_small")  
se <- mae[[1]]  
validate_se_assay_name(se, "RawTreated")
```

# Index

- \* **SE\_operators**
  - aggregate\_assay, [7](#)
  - demote\_fields, [19](#)
  - get\_MAE\_identifiers, [33](#)
  - identify\_unique\_se\_metadata\_fields, [40](#)
  - merge\_assay, [46](#)
  - merge\_metadata, [47](#)
  - merge\_SE, [48](#)
  - promote\_fields, [54](#)
  - SE\_metadata, [58](#)
  - split\_SE\_components, [60](#)
- \* **assay\_names**
  - get\_assay\_names, [24](#)
  - get\_combo\_assay\_names, [25](#)
  - get\_combo\_base\_assay\_names, [26](#)
  - get\_combo\_score\_assay\_names, [27](#)
  - get\_env\_assay\_names, [29](#)
- \* **combination\_data**
  - convert\_combo\_data\_to\_dt, [12](#)
  - convert\_combo\_field\_to\_assay, [13](#)
  - get\_combo\_col\_settings, [26](#)
  - get\_combo\_excess\_field\_names, [27](#)
  - get\_combo\_score\_field\_names, [28](#)
  - get\_iso\_colors, [33](#)
- \* **convert**
  - convert\_mae\_assay\_to\_dt, [13](#)
  - convert\_se\_assay\_to\_dt, [17](#)
  - df\_to\_bm\_assay, [20](#)
  - flatten, [22](#)
- \* **experiment**
  - get\_experiment\_groups, [31](#)
  - validate\_dimnames, [64](#)
  - validate\_MAE, [66](#)
  - validate\_SE, [67](#)
  - validate\_se\_assay\_name, [68](#)
- \* **fit\_curves**
  - .set\_invalid\_fit\_params, [6](#)
  - cap\_xc50, [10](#)
  - fit\_curves, [21](#)
  - logisticFit, [42](#)
  - predict\_conc\_from\_efficacy, [51](#)
  - predict\_efficacy\_from\_conc, [52](#)
  - set\_constant\_fit\_params, [57](#)
- \* **identifiers**
  - get\_default\_identifiers, [28](#)
  - get\_expect\_one\_identifiers, [30](#)
  - get\_identifiers\_dt, [31](#)
  - get\_ids\_synonyms, [32](#)
  - get\_required\_identifiers, [35](#)
  - headers, [38](#)
  - identifiers, [38](#)
  - prettify\_flat\_metrics, [53](#)
  - update\_env\_ids\_from\_mae, [63](#)
  - update\_ids\_synonyms, [63](#)
  - validate\_identifiers, [64](#)
- \* **internal**
  - .convert\_mae\_summary\_to\_json, [5](#)
  - .convert\_norm\_specific\_metrics, [5](#)
  - gDRutils-package, [4](#)
  - geometric\_mean, [24](#)
- \* **json\_convert**
  - convert\_mae\_to\_json, [15](#)
  - convert\_se\_to\_json, [18](#)
  - strip\_first\_and\_last\_char, [62](#)
  - validate\_mae\_with\_schema, [67](#)
- \* **json\_validate**
  - validate\_json, [65](#)
- \* **metadata\_management**
  - addClass, [6](#)
  - modifyData, [49](#)
- \* **package\_utils**
  - apply\_bumpy\_function, [8](#)
  - assert\_choices, [9](#)
  - average\_biological\_replicates\_dt, [9](#)
  - extend\_normalization\_type\_name, [20](#)
  - geometric\_mean, [24](#)

- get\_duplicated\_rows, 29
- get\_non\_empty\_assays, 34
- get\_synthetic\_data, 36
- is\_any\_exp\_empty, 40
- is\_exp\_empty, 41
- is\_mae\_empty, 42
- loop, 44
- MAEapply, 45
- mcolData, 46
- mrowData, 50
- shorten\_normalization\_type\_name, 59
- \* **standardize\_MAE**
  - get\_optional\_coldata\_fields, 34
  - get\_optional\_rowdata\_fields, 35
  - refine\_coldata, 55
  - refine\_rowdata, 55
  - rename\_bumpy, 56
  - rename\_DFrame, 57
  - standardize\_mae, 61
  - standardize\_se, 62
- \* **test\_helpers**
  - gen\_synthetic\_data, 23
  - get\_testdata, 36
  - get\_testdata\_codilution, 37
  - get\_testdata\_combo, 37
- .convert\_mae\_summary\_to\_json, 5
- .convert\_norm\_specific\_metrics, 5
- .set\_invalid\_fit\_params, 6
- addClass, 6
- aggregate\_assay, 7
- apply\_bumpy\_function, 8
- assert\_choices, 9
- average\_biological\_replicates\_dt, 9
- bplapply, 44
- cap\_xc50, 10
- convert\_colData\_to\_json, 11
- convert\_combo\_data\_to\_dt, 12
- convert\_combo\_field\_to\_assay, 13
- convert\_mae\_assay\_to\_dt, 13
- convert\_mae\_to\_json, 15
- convert\_metadata\_to\_json, 15
- convert\_rowData\_to\_json, 16
- convert\_se\_assay\_to\_dt, 17
- convert\_se\_to\_json, 18
- demote\_fields, 19
- df\_to\_bm\_assay, 20
- extend\_normalization\_type\_name, 20
- fit\_curves, 21
- flatten, 22
- gDRutils (gDRutils-package), 4
- gDRutils-package, 4
- gen\_synthetic\_data, 23
- geometric\_mean, 24
- get\_assay\_names, 24
- get\_combo\_assay\_names, 25
- get\_combo\_base\_assay\_names, 26
- get\_combo\_col\_settings, 26
- get\_combo\_excess\_field\_names, 27
- get\_combo\_score\_assay\_names, 27
- get\_combo\_score\_field\_names, 28
- get\_default\_identifiers, 28
- get\_duplicated\_rows, 29
- get\_env\_assay\_names, 29
- get\_env\_identifiers (identifiers), 38
- get\_expect\_one\_identifiers, 30
- get\_experiment\_groups, 31
- get\_header (headers), 38
- get\_identifiers\_dt, 31
- get\_ids\_synonyms, 32
- get\_iso\_colors, 33
- get\_MAE\_identifiers, 33
- get\_non\_empty\_assays, 34
- get\_optional\_coldata\_fields, 34
- get\_optional\_rowdata\_fields, 35
- get\_prettified\_identifiers (identifiers), 38
- get\_required\_identifiers, 35
- get\_SE\_experiment\_metadata (SE\_metadata), 58
- get\_SE\_experiment\_raw\_data (SE\_metadata), 58
- get\_SE\_fit\_parameters (SE\_metadata), 58
- get\_SE\_identifiers (SE\_metadata), 58
- get\_SE\_keys (SE\_metadata), 58
- get\_SE\_processing\_metadata (SE\_metadata), 58
- get\_synthetic\_data, 36
- get\_testdata, 36
- get\_testdata\_codilution, 37
- get\_testdata\_combo, 37
- headers, 38

identifiers, 38  
identify\_unique\_se\_metadata\_fields, 40  
is\_any\_exp\_empty, 40  
is\_exp\_empty, 41  
is\_mae\_empty, 42

lapply, 44  
logisticFit, 42  
loop, 44

MAEapply, 45  
mcolData, 46  
merge\_assay, 46  
merge\_metadata, 47  
merge\_SE, 48  
modifyData, 49  
mrowData, 50  
MultiAssayExperiment, 14

predict\_conc\_from\_efficacy, 51  
predict\_efficacy\_from\_conc, 52  
prettify\_flat\_metrics, 53  
promote\_fields, 54

refine\_coldata, 55  
refine\_rowdata, 55  
rename\_bumpy, 56  
rename\_DFrame, 57  
reset\_env\_identifiers (identifiers), 38

SE\_metadata, 58  
set\_constant\_fit\_params, 57  
set\_env\_identifier (identifiers), 38  
set\_SE\_experiment\_metadata  
(SE\_metadata), 58  
set\_SE\_experiment\_raw\_data  
(SE\_metadata), 58  
set\_SE\_fit\_parameters (SE\_metadata), 58  
set\_SE\_identifiers (SE\_metadata), 58  
set\_SE\_keys (SE\_metadata), 58  
set\_SE\_processing\_metadata  
(SE\_metadata), 58  
shorten\_normalization\_type\_name, 59  
split\_SE\_components, 60  
standardize\_mae, 61  
standardize\_se, 62  
strip\_first\_and\_last\_char, 62  
SummarizedExperiment, 13, 17, 41, 59–61,  
69

update\_env\_idfs\_from\_mae, 63  
update\_idfs\_synonyms, 63

validate\_dimnames, 64  
validate\_identifiers, 64  
validate\_json, 65  
validate\_MAE, 66  
validate\_mae\_with\_schema, 67  
validate\_SE, 67  
validate\_se\_assay\_name, 68